Solving a business problem with a Message Queue, DebuggableService and XML Serialization

Introduction

When working with a client in the hardware maintenance business, I developed an efficient and robust solution to solve a specific business problem. This blog will discuss the business requirement and how I solved it with a simple solution using a message queue, DebuggableService, and XML Serialization.

Business Requirements

A few months ago, I received a requirement to add **shipping label printing functionality** to a customer's web application. The customer had several facilities throughout the world and each facility had many label printers that are used by other applications to print mailing labels.

The customer's web application was implemented in ASP.NET and Web API 2, using C#, knockoutJS, and SQL Server. The **shipping label printing functionality** would need to be added to the current application.

A label had to be printed after successfully receiving an item, and attached to the item, before it was moved to the warehouse.

There are over 50 label types with different formats and content that can be generated, each with their own specific data requirements.

Solution

As there are potentially hundreds of users using the application at any time, I did not want to trigger a label print directly from the web application, which may have put stress on network and the label printers. As such, much thought went into the design. It was determined to decouple the request for a label to be printed through the application and the actual printing of the label. As such, I decided to use a message queue to capture label printing requests generated when an order was received.

Upon successfully order receipt, the application would write a message to the message queue. The message would include printer name, number of labels to print, label type, label template location (on the server), label specific data, and template name. To do the actual label printing, I implemented a windows service to read and process messages from the message queue and to format and send print requests to the label printers.

To make the development of the windows service efficiently, I downloaded the **DebuggableService** Project Template from the Microsoft Visual Studio Marketplace. This project template made it easy to both debug the service as a console project, and install the service as a windows service.

As there was to be only one global windows service to handle queue messages, I opted to use a SQL Server Table as the message queue. The service would continually poll the queue for messages, read any and all messages, process the messages, delete the messages after successful processing, log processing result to an application log, and then resume message polling.

I could have opted to use an actual message queue but would have had to create the queue which would added more complexity, and pieces to the application, with no strong benefit.

The schema for the queue table (LabelQueue) is listed in figure 1, below:

	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	CreateDate	datetime	☐	(getdate())
	PrinterName	nvarchar(256)	☐	
	NumberOfLabelsToPrint	int	☐	
	LabelType	nvarchar(50)	☐	
	TemplateFileLocation	nvarchar(MAX)	☐	
	ReceiptItemViewModel	nvarchar(MAX)	☐	
	ReportTemplateName	nvarchar(256)	☐	
	PrintError	bit	☐	((0))

Figure 1 LabelQueue Schema

In order to minimize change management regarding the LabelQueue table, I opted to pass all label configuration messages as XML within the ReceiptItemViewModel column. This prevents the need to modify the table if any label content is modified or if new labels are added.

I opted to use XML Serialization to format, write and read the information within the ReceiptItemViewModel column.

To format the information correctly as XML, I used the following code:

```
var flattenedModel = XmlUtilities.SerializeToXml<ReceiptItemViewModel>(model)
```

which calls the following .NET API code to serialize the class to XML:

```
public static string SerializeToXml<T>(object myObject)
{
    var stringwriter = new System.IO.StringWriter();
    var serializer = new XmlSerializer(typeof(T));
    serializer.Serialize(stringwriter, myObject);
    return stringwriter.ToString();
}
```

To read the information within the Windows Service, I used the following code:

```
var receiptItemViewModel =
XmlUtilities.LoadFromXmlString<ReceiptItemViewModel>(myXml)
 as ReceiptItemViewModel
```

which calls the following .NET API code to deserialize the class back to an object:

```
public static object LoadFromXmlString<T>(string xmlText)
```

```
{
    var stringReader = new System.IO.StringReader(xmlText);
    var serializer = new XmlSerializer(typeof(T));
    return serializer.Deserialize(stringReader);
}
```

Another approach would have been to directly create a data model to hold individual elements of the label in specific data table columns. However, this approach would lead to table modifications, if the label data requirements changed in the future.

Conclusion

This blog discussed a business requirement to print shipping labels from an existing application, and how I implemented it with a simple solution using a message queue implemented as a table in SQL Server, DebuggableService a project template obtained from the Microsoft Visual Studio Marketplace, and XML Serialization, part of the C# API. The solution effectively decoupled the request and processing of the label printing functionality and minimized the potential for database modification should the label data be modified in the future.

Author

John F. Gnazzo, PE MBA PMP and Microsoft MVP (Visual Studio and Development Technologies 2016-2017)

Gnazzo's role at SCS is to provide architecture, development, and technical leadership expertise across the full project lifecycle to SCS staff, clientele and community. Gnazzo is also committed to being a leader in the implementation of enterprise business solutions comprising of contemporary web, desktop and mobile technologies, in the Microsoft technology space.

In addition, he authors a monthly world-wide coding challenge to engage developers to solve real-world technical problems, and routinely shares his first-hand experience implementing technology via blogs, comments and case studies.

John also volunteers his time speaking at symposia, and mentoring students and the community regarding technology, consulting and problem solving.

Gnazzo is a registered professional engineer with a Minnesota license, a certified project management professional, and a Six Sigma Green Belt, and is working towards his Microsoft Certified Solution Developer certification.